

# Component Based Architecture Supplementing Service Oriented Architectures

**Badri Sriraman**  
**Lead IT Architect**  
**(Unisys)**

**Rakesh Radhakrishnan**  
**Enterprise IT Architect**  
**(Sun Microsystems)**

**March, 2005**



## Table of Contents

Introduction & Overview.....	Page 3
Value Proposition of Component based development for Services.....	Page 8
Cohesive and serviceable components engender durable services.....	Page 10
Wide component container capabilities provides configurable QoS.....	Page 11
Distributing Components for Dynamic Service interactions.....	Page 12
Code Mobility for Service Elasticity .....	Page 13
Functional decomposition for Service Align-ability.....	Page 14
Conclusion.....	Page 15

## Introduction and Overview

So far in this series of papers on Service Oriented Architectures the authors have highlighted the significance of Model Driven Architectures and Event Driven Architectures to Service Oriented Architectures, in conjunction with an Identity enabled Networks. This paper attempts to highlight the significance and value proposition of leveraging the fundamental principle's and concepts behind "Component Based Architectures" (CBA) for "Service Oriented Architectures" (SOA), for XML based Services. This paper highlights the fact that CBA supplements SOA – and CBA, EDA, SOA –all approaches compliment and support User Centric Agile Solutions.

In **Service Oriented Architectures (SOA)**, an Enterprise's Architecture is developed in a Service Driven approach, where each service is autonomous and can be considered to be a Service Building Block (SBB). Using an analogy between the concept of service and a business process, in SOA, loosely coupled SBB are orchestrated into business processes that support customer goals and/or organization's business goals. More than just a component of reusable code, a SBB becomes part of a running program that can be invoked by a client without having to incorporate the code itself. A SBB by definition is reusable and replaceable, that is, one SBB's service is reused again and again by other services for the functionality it provides and SBB's service (provider implementation) can be replaced by another (another providers implementation). In SOA – the SBB's themselves can be categorized into basic/foundation services, management services, security services, business services, portal services, etc. It should also be noted that a SBB is offering a specific functionality for an Enterprise and transcends projects, i.e., a Digital Rights Management Service (DRM) as a SBB is only implemented once in an enterprise architecture and can be reused across projects that deal with delivering content, services, multi-media, etc. In SOA the communication flow is closed to new unforeseen inputs once the communication flow has started, i.e., they are typically well defined and the boundaries well set.

**Component Based Architecture**, the basis for distributed component architecture embraces mechanisms and techniques for developing coarse yet reusable business/technical implementation units that is environment/container aware, which decomposes a service of SBB, into multiple plug-able and distributable parts associated with presentation logic, business logic, resource access logic, integration logic, network event logic, security logic and more. A component based approach evolves from object oriented design principles (encapsulation, polymorphism, inheritance, etc.), and leverages a foundation set of infrastructure technologies (based on J2EE/J2ME/JAIN or .NET/OSA/Parlay) for seamless integration. Component Based Architecture (CBA) fundamentally modularizes large scale (monolithic) systems into multiple coarse-grained, durable and reusable technical units that could be implemented by multiple vendors yet integrated into a larger enterprise system. CBA typically engenders designing the internals of these coarse-grained, business-aligned component boundaries, through finer-grained object-orientation. Lately the capabilities to instrument components with service contracts around security and SLA's, has made component based design even more powerful.

**Event Driven Architecture**, embraces mechanisms for coordinating the callers and providers of service, producers and consumers of data, sensors and responders of software events with variable level of communication coupling, with variable spectrum of message correlation and with variable options to deliver quality of service. EDA engenders a network that listens to thousands of incoming events from different sources, wherein complex event processing results in intended system response. EDA supports dynamic, parallel, asynchronous flows of messages and hence reacts to external inputs that can be unpredictable in nature. For example EDA approach can enable a caller invoke a provider SBB using events without knowing who provides it or what address the provider uses, what choice amongst multiple providers exists, while load balancing across them and selecting amongst these providers with varying levels of qualities of service based on the caller's requirements. Meanwhile the same software event that represents the request or the events generated by the service in response can be of interest to other SBB in the network, opening a channel for value-add to the customer and business. An EDA can coordinate synchronously or asynchronously between software endpoints, and possibly provide both synchronous and asynchronous access between the same participants. In EDA, simultaneous streams of execution can run independently of each other to fulfill a customer request or system responsibility, typically an event bus serves as a platform to manage integration and/or choreograph a larger process.

In **Model Driven Architecture**, modeling an enterprises system (data, systems, and model of your data model) in conjunction with meta-data, which keeps a record of an enterprise's architecture in-terms of data, information, application, services, technology and implementation (platform specifics) is the basis. There are three basic tools used in an MDA as defined by OMG (Object Management Group) – (see <http://www.omg.org/mda>): **Meta-Object Facility (MOF)**- The MOF defines a standard repository for meta-models and, therefore, models (since a meta-model is just a special case of a model). **XML Meta-Data Interchange (XMI)** - XML defines an XML-based interchange format for UML meta-models and models; by standardizing XML document formats and DTD (document type definitions). In so doing, it also defines a mapping from UML to XML. **Common Warehouse Meta-model (CWM)** - The CWM standardizes a complete, comprehensive meta-model that enables data mining across database boundaries at an enterprise and goes well beyond. Like a UML profile but in data space instead of application space, it forms the MDA mapping to database schemas.

### **An example Scenario where Components supplement Services:**

Since this example is based on a sample scenario that leverages almost all of the Java technologies a brief introduction of the same is required (for more details on each of these technologies visit <http://java.sun.com>, also there are books on each of these topics too). The entire spectrum of Java Technology Platform includes;

- **J2EE**, Java 2 Platform, Enterprise Edition (J2EE) defines the standard for developing component-based multi-tier enterprise applications.
- **J2ME**, The Micro Edition of the Java 2 Platform provides an application environment that specifically addresses the needs of commodities in the vast and rapidly growing consumer and embedded space, including mobile phones, pagers, personal digital assistants, set-top boxes, and vehicle telemetric systems.
- **Java Web services** are Web-based enterprise applications that use opens, XML-based standards and transport protocols to exchange data with calling clients. Java 2 Platform, Enterprise Edition (J2EE) provides the APIs and tools you need to create and deploy interoperable Web services and clients. Java based Web Services are organized into:
  - Java Web Services Developer Pack (Java WSDP)
  - Java API for XML-Based RPC (JAX-RPC)
  - Java API for XML Registries (JAXR)
  - Java API for XML Processing (JAXP)
  - Java Architecture for XML Binding (JAXB)
  - SOAP with Attachments API for Java (SAAJ)
- **Java Card** technology provides a secure environment for applications that run on smart cards and other devices with very limited memory and processing capabilities.
- **Jini** network technology, which includes JavaSpaces technology and Jini extensible remote invocation (Jini ERI), is an open architecture that enables you to create network-centric services--whether implemented in hardware or software--that are highly adaptive to change.
- **JXTA** technology is a set of open protocols that enable any connected device on the network, ranging from cell phones and wireless PDAs to PCs and servers, to communicate and collaborate in a P2P manner.
- **OSS through Java Initiative** ("OSS" stands for "Operations Support Systems") produces a standard set of Java technology-based APIs to jump-start the implementation of end-to-end services on next-generation wireless networks, and leverage the convergence of telecommunications and Internet-based solutions.
- The **JAIN** initiative has defined a set of Java technology APIs that enable the rapid development of Java based next generation communications products and services for the Java platform.
- **Java Dynamic Management Kit** (Java DMK) is a Java technology based toolkit that allows developers to rapidly create smart agents based on the Java Management Extensions (JMX) specification. The power of the JMX framework is that it supports multiple protocol access to management information residing in the agent.
- The **JMI** specification enables the implementation of a dynamic, platform-independent infrastructure to manage the creation, storage, access, discovery, and exchange of metadata. JMI is based on the Meta Object Facility (MOF) specification from the Object Management Group (OMG), an industry-endorsed standard for metadata management.
- And few more (such as JBI, JTAPI and Java Media Extensions)

Authors Alan Brown, Simon Johnston and Kevin Kelly in a Rational Software whitepaper out in 2002, refer to even C++ as a monolithic application development environment that is not conducive for true component based architectures. There are not many programming paradigms that exist today that fully support Component Based Architectures that leads distributed components under SOA. Hence this example is based on a CBA solution that is built on the Java Technology Platform (that include J2EE/J2ME and more) to show how it supplements SOA.

The example cited in the EDA augmenting SOA paper is a sample scenario that's made possible with this spectrum of Java technologies. This is not a trivial scenario and is quite complex in its nature due to multiple client devices involved, multiple access networks involved, multiple services involved, user actions-reactions are taken into account, etc. Let's examine this scenario further.

Today within Telco environments services such as Identity, Location, Presence, Weather, have been implemented as a re-usable service, i.e., the location logic running in a Location service can be re-used for many other services – such as location based weather report, location based mapping, location based retail list, etc. Now as an end user I travel to a particular city – such as Seattle for an important meeting the next morning. I setup my alarm for 6:00am on my cell phone and along with my alarm; I receive a short weather report and a Doppler image for the specific location that I'm in. Majority of this scenario is made possible due to a Service Oriented Architecture (SOAP/XML). Now by adding the concepts behind Event Driven Architecture, this scenario can get even more interesting. Lets say due to time difference (5:30PST is actually 8:30EST) I actually wake up before the alarm goes off, and access/login to a TV/STB. With my presence in the network, now, I only receive a weather report/Doppler image on my cell phone and not the alarm, since an event gets triggered the minute I login stating that I'm present and hence awake (canceling the wake-up call). This could be taken one step further, when I not only access TV/STB services, but I actually view a weather channel report. Based on my consumption of this service another event takes place where the weather report/Doppler image along with the wake up call, all gets cancelled. (This is user centric!!). At the same time the flight that was scheduled to depart at 2:00pm that afternoon from Seattle is delayed due to storm in my destination city (Washington DC) and an event is triggered to send me a message notifying the same, with an alternate schedule.

- The Weather Service is itself a simple Web Service, that has a highly explicit and streamline interface (SOAP/XML) so that other services can leverage it as a reusable SBB – this is the basis and a good example of a SOA where location service logic refers the location info to obtain location based weather information. Hence here, both J2EE and Java Web Services are used as a component based application environment – that executed multiple modular logic in the form of EJB's and MDB's.
- The alarm setup on the cell phone that is an identity enabled device is based on the J2ME technology and the components involved are midlets, servlets, etc. J2ME with its support for protocols such as SIP, OCAP and WAP – runs on multiple identity enabled devices such as cell phones and STB/TV's.
- Logging into the Cable Network via a STB/TV triggers the presence service in the network and event based technology (such as JAIN) is used to cancel a telecom service (such as alarm/notification), and web service (such as the location based weather service). Components involved here include Xlets and JAIN SLEE components.
- One add-on to this scenario might include leveraging a P2P service between the STB/TV and the Cell Phone (connected to the network), where in the weather report is saved on both device for the day. Jxta components are executed in the network for this extended p2p service to work.

- In this scenario, OSS though Java initiative is leveraged, such as billing mediation, to ensure appropriate billing logic gets executed and mediation occurs between multiple access network providers and the core network provider, as well as the Identity Service Provider and other Service Providers.
- The alarm service in itself could be rendering user centric alarm tones that are played by IP media services, using XML dialects such as MSML and MOML (two related markup languages related to media services). Using techniques from JMI and its support to OMG's XMI, the underlying dialect interchanges can be handled through an XMI repository. Many XML components are involved here including XMI components and other XML components.

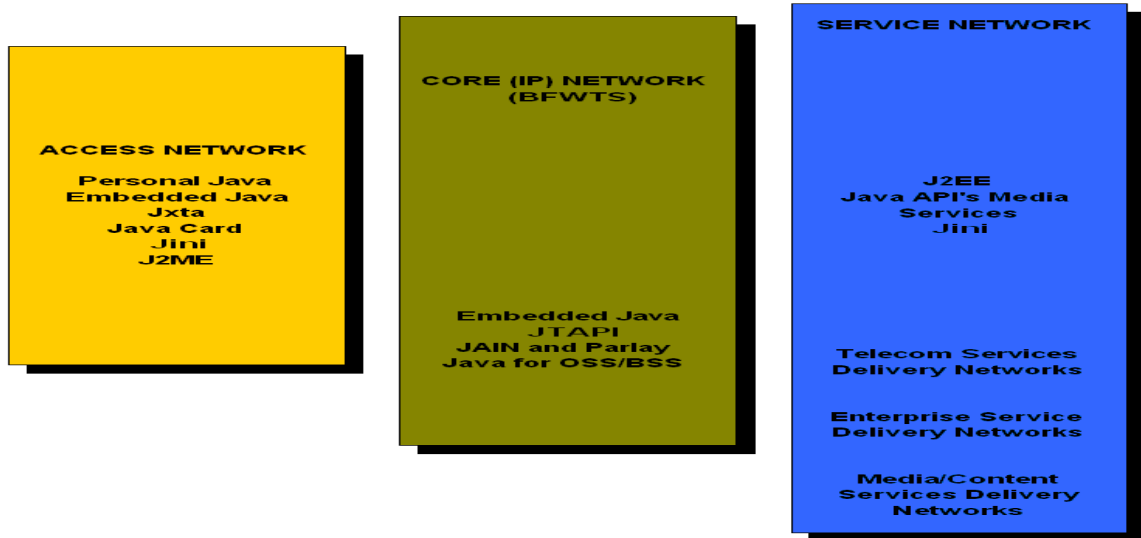


Figure 1: Java Technology Platform as a CBA

This is an example that highlights the value-proposition that components bring to services. It also highlights the significance of MDA and EDA, along with CBA to enable, augment and supplement SOA in various manners. Jini, J-DMK and component instrumentation also play a significant role here to SLA and OLA around services are well managed within the Service Networks.

What needs to be noted is that these Java Technologies work in conjunction with IP initiatives such as MobileIP and IPSec as well as XML technologies including SAML and XKMS.

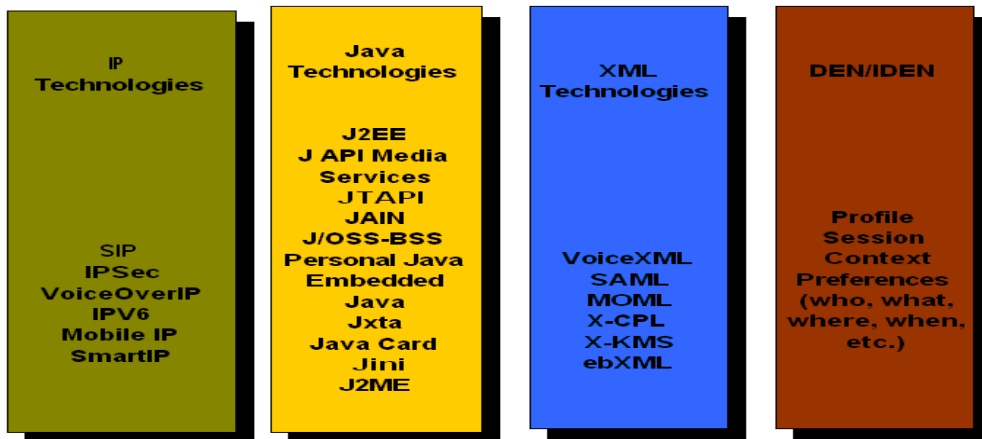


Figure 2: Java Technology Platform + IP, XML and Directory Technologies

This paper is the fourth in a series, that includes;

- Model Driven Architecture enabling Service Oriented Architectures
- Event Driven Architecture augmenting Service Oriented Architectures
- Identity Centric Architectures (IDEN) for Service Oriented Architectures
- Component Based Architecture supplementing Service Oriented Architectures
- Utility Compute Architectures supporting Service Oriented Architectures

All five papers describe the power in SOA when other conceptual architecture patterns associated with MDA, EDA, CBA and UCA are blended with SOA. With an identity enabled end-to-end environment, it adds more meaning to the notion of Service on Demand and True Service Mobility – anywhere, anytime, any network, and any device. The first paper highlighted how MDA enables SOA, following that the second paper highlighted how EDA, in conjunction with MDA, augments SOA. Along the same lines the notion of end-to-end Identity enabled Environment, was discussed, to highlight the Core Service Building Block of Service Oriented Architectures. This paper will address the value-proposition and supplementing capabilities brought to SOA by CBA, in conjunction with EDA and MDA. The last in the series will highlight the alignment of SOA with UCA and how CBA, MDA and EDA with IDEN pave the way to do so.

**Value Proposition of Components (CBA) for Services (SOA):**

If we assume a world where SOA and its fundamental principles around streamlined/simplistic interface design for seamless integration between other services will suffice, then any monolithic application, that is not modular, not-distributable, not-instrument-able, not-extensible to multiple clients, etc., wrapped with a XML interface for service interaction is good enough. However taking this approach with programming paradigms that do not support a component based development will lead to non-granular set of services that will have disruptive effect on the entire application on change, that are difficult to distributed and be accessed from multiple networks/devices, that are not instrumented with a boundary for service management and more. What CBA offers in terms of supplementing SOA is exactly addressing these set of issues by taking into account the need for componentization in design and implementation that support capabilities to;

- Evolve right-grained implementation units
- Remain goal-focused with infrastructure services exported to containers
- Build cohesive, reusable and serviceable units
- Build mobile and distributed logic
- Self-describing components to support instrumentation
- Aggregate component assembly that deliver coarse services
- Mitigate disruption due to change
- And more.

CBA also aligns well with EDA, MDA and SOA from a holistic approach, wherein, some of the distributed components are actually events in the network, and MDA aids in the space of component instrumentation, to allow for the alignment of Software Architectures with IT Architectures. This is explained further in detail in this paper.

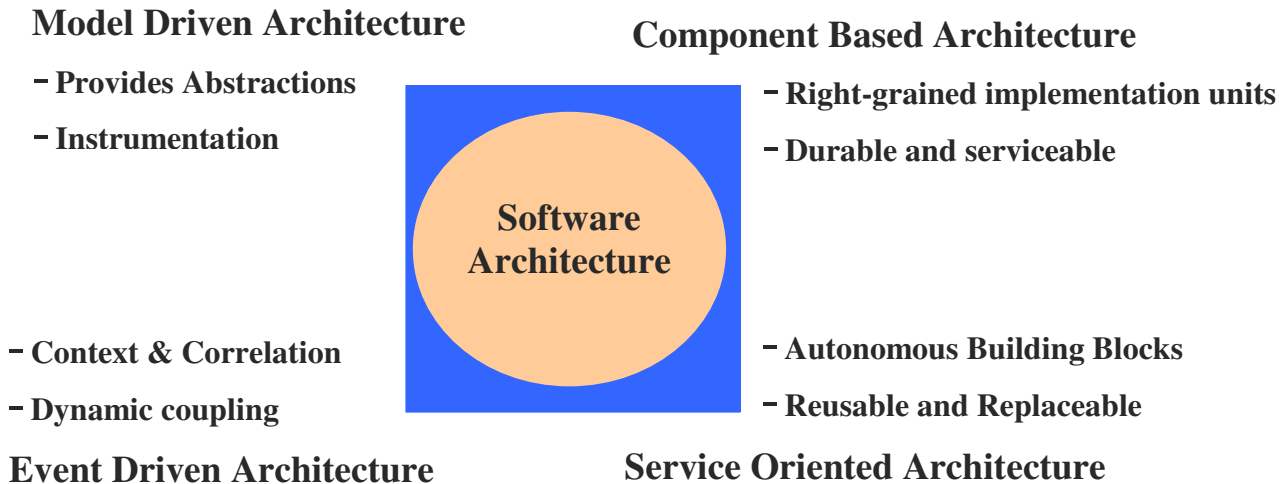


Figure 3: Quadrangle View of Architectural Strategies

There are several perspectives from different architectural stakeholder viewpoints where componentizing add value to services, and they are;

- ***Cohesive and serviceable components engender durable services***
  - ***Impact-free localized changes enable continuous re-factoring***
- ***Wide component container capabilities provides configurable QoS***
  - ***Instrument-ing components for higher Service Quality***
- ***Distributing components for dynamic service interactions***
- ***Code Mobility for Service Elasticity***
- ***Functional Decomposition of Services for Align-ability***

### **Cohesive and serviceable components engender durable services**

This perspective acknowledges that well discovered components supplement service durability and there by architecture longevity. Typically, well analyzed requirements lead to key domain abstractions that form a basis for a cohesive coarse analysis objects. These have a direct correlation to service contracts between subsystems that for the application. In component based architecture you discover your key components and organize them around these key abstractions and subsystems. Requirement refinements will rightly translate into component specification around these components, making cohesive components which expose services in a consistent manner. In a many cases in real-world requirements evolve and scope management will push realization to a different time. And in all cases the application is expected to have new system requirements to react to changing business needs. Components built around key abstractions will have the least specification impact and will be able to expose new services around it. This allows for following benefits:

- Subsystems that have dependency to this existing services survive changes
- New subsystems will develop non-invasively

This extends life-span of overall application and that validates the architecture bearing of a component based architecture that supplements service-orientation.

This perspective also acknowledges that software services always require changes in one form or another and its important to ensure impact-free localized changes enables continuous re-factoring. In some cases the business logic might change, in other the presentation logic/rendering might be extended to a new devise and protocol, in some case the integration logic might change (i.e., from a pure messaging and publish/subscribe type to a transactional-messaging and publish/subscribe/verify type) and so on. With CBA one of the core principles is to not only modularize big chunks of business logic into granular service that constantly interact, but also to modularize into tired logic, and abstracting between tiers to a point where impact of changes in one tier does impact another. A simple yet powerful example often used in CBA is the MVC Pattern that allows for the decoupling of presentation logic from the business and logic and introduces layer between the two, so that extension of the presentation logic to multiple devices (fro e.g., from a J2ME/WAP phone to a J2ME/OCAP STB), is seamless and does not require changes in other tiers.

This modular approach in conjunction with Service Container capabilities address real-time changes that can be made to a service implementation without impacting the entire system. For example after running end-to-end tests of a component the same can be provisioned into an environment with automated tools (e.g., Service Provisioning tools for J2EE environments) with minimal or no downtime to the service itself.

#### **Model Driven Architecture**

#### **Component Based Architecture**



- Cohesive and serviceable
- Durable and extendible
- Modularizing Services into Components
- Tiered Logic with its respective Component
- Real-time changes to components

#### **Event Driven Architecture**

#### **Service Oriented Architecture**

**Figure 4: CBA with durable components supplementing SOA**

### Wide component container capabilities provides configurable QoS

This perspective presents, how, many service quality attributes can be easily implemented using components that engage containers. In Component Based architecture, rightly deployed components engage container services that deliver enterprise services and many other orthogonal services that make appropriate sense in the environment and context of its execution. In J2EE environment we can engage complex services that manage pooling and life-cycle by mere configuration. Similarly, in .Net environment we can use simple meta-data attributes to define transaction boundaries around services. This allows for quick and reliable development of components that constitute services which require many of these orthogonal services like transaction, resource management, security and life-cycle management. This allows client that call these services to be oblivious of the service delivery mechanism providing a clean separation of concern, but yet declaratively engage these QoS facilities. Container-managed enterprise services have become standard aspect in most design strategies. It has allowed architects to focus on the business problem.

A similar perspective acknowledges Richard Hubert’s theme on his book titled “Convergent Architectures” –essentially covering the topic of instrument-ing components for higher service quality. The fundamental and strategic idea here is the notion of building software systems with both service function in mind (such as CRM, SCM, travel services, etc.) and the service management functions in mind (i.e., the support requirements of the service post deployment). Typically one role is left to software and application architects while the other role is left to the IT/Infrastructure Architects. In his book what is being proposed with extensive examples using the Java Technology Platform is the convergence of this space wherein, the software development upfront takes into account all service quality requirements and instruments the components with telemetry and related alignment capabilities to ensure successful delivery and support aspects of such business services. This is breakaway from traditional development practices primarily made possible with CBA supplementing SOA. From the overall system perspective delivering services to user requests can involve choreographing a large service calls to prepare a single simple response, immediately or over a period of time repeatedly or a complex response to be delivered over multiple interactions, user sessions and in multiple formats. Such service rendering is managed and delivered appropriately only when a service’s components are;

- Instrumented with telemetry (recognize varying conditions)
- Instrumented with code to respond under varying conditions

**Model Driven Architecture**

**Component Based Architecture**



- Container Management
- Ease of development
- Improved QoS
- Instrumented Components
- End-to-end Service Quality

**Event Driven Architecture**

**Service Oriented Architecture**

Figure 5: CBA with container managed components supplementing SOA

## Distributing Components for Dynamic Service interactions

As discussed in the above example there are multiple components within and around a service that are typically distributed across multiple service networks (intra and inter service networks) and between the access networks/devices (Cable, Wifi, 3G, etc.) and the Core Networks (events). This makes a service truly mobile, in one sense, since accessing the services (residing in different enterprise networks and service provider networks) from a device agnostic and service location agnostic manner is made possible based on components that embed this intelligence in devices and access networks. From a SOA perspective a single EJB or MDB that supports a streamlined interface (SOAP/XML) and a publicly registry (a service available for use) makes it a “web service”. However the context based delivery of this service (built as a EJB or MDB) can benefit from many external components, components from Xlets, Midlets, Applets, JAIN/SLEE components, Jxta and more, (that can invoke these other service components) to ensure dynamic service interaction with the User, within given SLA/OLA boundaries is possible. Such service interaction needs to be captured and billed for accordingly, using other related components that are part of the overall boundaries of the service’s architecture.

Modularized Services as Components distributed in the network ensures;

- Service Mobility aspects of SOA (going beyond user and device mobility): where access to a service is made access network agnostic and access device agnostic (e.g., IPTV – broad cast services accessed from a STB/TV over a Cable Network and a Cell Phone of 3/4G Networks or IP-Music from XM-Radio receiver over a Satellite network or from a Laptop over Wifi Network, etc.).
- Service on Demand aspects of SOA (contextual on-demand access to services): where a Service is delivered to user based on his or her profile and preference (e.g., profile requesting a wakeup alarm every morning, preference requesting a notification when a book is available to be shipped, etc.).

**Model Driven Architecture**

**Component Based Architecture**



- Distributed Components
- Dynamic Service and Component Interactions

**Event Driven Architecture**

**Service Oriented Architecture**

**Figure 6: CBA with distributable components supplementing SOA**

### Code Mobility for Service Elasticity

This perspective takes into account the mobility of code when Services are developed based on components and what it means to service elasticity. Given the fact that SOA makes service boundary-less i.e., traditionally when we implemented an HR system for an enterprise the boundaries were clear in terms of the user population (3000 user, etc.), time of access, within an enterprise WAN, etc. However when a payroll system is deployed as a web service that can be subscribed to on the web and the user population could reach the millions from anywhere in the globe at any point in time, with extreme fluctuations from a systemic quality requirements perspective. This implies that the service in order to meet SLA's and to address QOS issues has to possess principles of elasticity. With monolithic code with rigid deployments there are limits to its support boundaries, however with mobile code, and real-time deployment of components to containers, true elasticity can be achieved. In one extreme the service components could be running on two containers and in another minute on 200 containers distributed in the service network. Componentizing Services ensures alignment of Application Service Containers (such as J2EE Containers and J-SLEE Containers) to System Service Containers (such as Solaris Containers and Linux VM ware). Elasticity is achieved while taking into account the affinity requirements of certain software components to certain hardware components (e.g., components that handle cryptography isolated to hardware with crypto accelerators).

**Model Driven Architecture**

**Component Based Architecture**

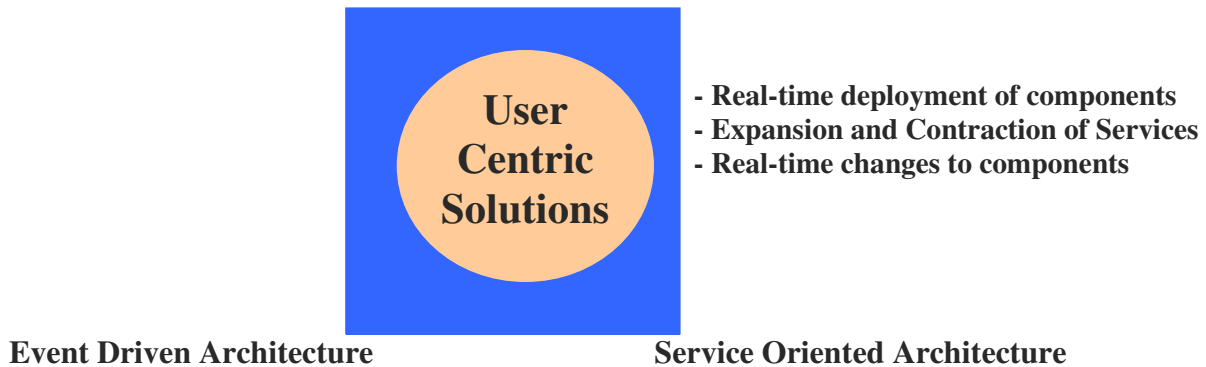


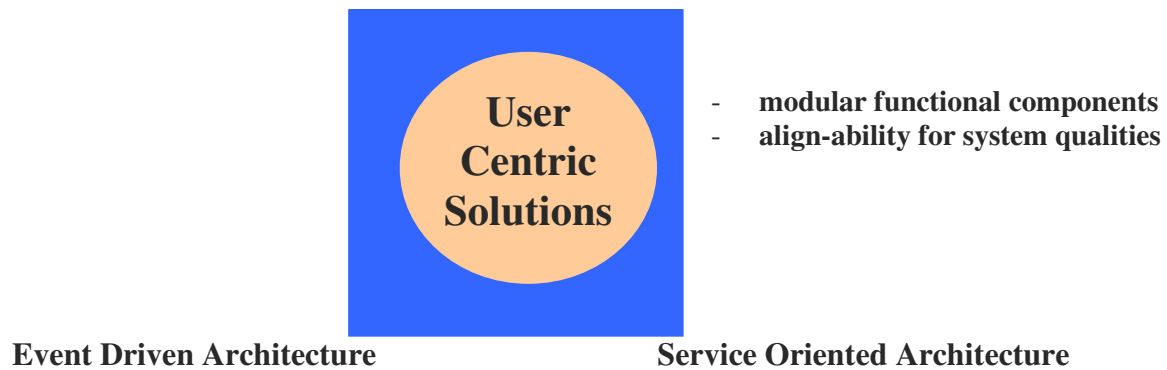
Figure 7: Code Mobility for Service Elasticity with Affinity

### Functional decomposition of Service for Align-ability

This perspective takes into account the functional decomposition that takes place when a service is built based on modular components and its impact on align-ability of a service to constant changes in requirements. This perspective can be best understood with an example. If a Service is built in a monolithic manner wherein, its security capabilities are built-in hard coded features of the system, then align-ability of security concerns suffers. That is changes to any security mechanisms will involve changes to the system as a whole. Whereas, when the service is built based on external security components that it leverages on a need basis, then security concerns are considered to be more align-able. Instead of leveraging a built-in custom code that is tightly coupled with the service logic, one can leverage external security mechanisms associated with security, such as, JAAS (java authentication and authorization services) and JCE (java cryptography extensions). Within JAAS there is support for a spectrum of authentication mechanisms from weak authentication to standard authentication, all the way to strong authentication, that a Service built-on component based design can leverage on an ongoing basis. Similarly JCE offers a spectrum of encryption mechanisms with differing algorithms, for ranges in the strength of encryption and resource consumption. Service when leveraging external JCE components can be aligned in accordance with its encryption needs at given points in time. This fundamental benefit associated with component based design is applicable for multiple systemic quality concerns ranging from security, to availability to scalability and manageability. Also modular functional decomposition works in sync with code-mobility, component instrumentation and the distributable nature of components.

**Model Driven Architecture**

**Component Based Architecture**



**Figure 7: Functional decomposition for Service Align-ability**

## **Conclusion**

Embraces all the five areas of synergies discussed between Components based Architectures for SOA in this paper, leads to more Agile Software Architecture that can accommodate change more gracefully, than when compared to other approaches.

Cohesive and serviceable components that engender durable services hosted in component containers (providing configurable QoS) at the same time distributing components for dynamic service interactions (between many/types of Containers) plus Instrument-ing components for higher Service Quality along with Impact-free localized changes (that enable continuous re-factoring) all lead to the possibility of addressing changes more gracefully, whether in the functional or non-functional requirements. Leveraging a DEN/IDEN (directory/identity enabled network) and delivering via an Event Execution environment then the Enterprise (Technology) Architecture is set to achieve Totally Agile Solutions that uses end-to-end integration with straight through processing. All these approaches leads to the strategic benefit associated with CBA supplementing SOA, MDA enabling SOA and EDA augmenting SOA, especially in a web services world, where Services are delivered on demand.

***Copyrights***

**©2005 Sun Microsystems, Inc. All rights reserved.**

**Sun Attribution Language:**

**Sun, Sun Microsystems, the Sun Logo, Sun Enterprise, Java Enterprise Systems and Solaris are trademarks or registered trademarks of Sun Microsystems, Inc. in the United States and other Countries.**